Apache Accumulo 1.4 & 1.5
Features

Inaugural Accumulo DC Meetup

Keith Turner
What is Accumulo?

A re-implementation of Big Table

- Distributed Database
- Does not support SQL
- Data is arranged by
  - Row
  - Column
  - Time
- No need to declare columns, types
Massively Distributed
# Key Structure

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td>Value</td>
</tr>
<tr>
<td>Column Family</td>
<td>Value</td>
</tr>
<tr>
<td>Column Qualifier</td>
<td>Value</td>
</tr>
<tr>
<td>Column Visibility</td>
<td>Value</td>
</tr>
<tr>
<td>Time Stamp</td>
<td>Value</td>
</tr>
<tr>
<td>Primary Partitioning Component</td>
<td>Value</td>
</tr>
<tr>
<td>Secondary Partitioning Component</td>
<td>Value</td>
</tr>
<tr>
<td>Uniqueness Control</td>
<td>Value</td>
</tr>
<tr>
<td>Access Control</td>
<td>Value</td>
</tr>
<tr>
<td>Versioning Control</td>
<td>Value</td>
</tr>
</tbody>
</table>
Tablet Organization

Well-Known Location (zookeeper)

Root Tablet
\(-\infty \text{ to } \infty\)

Metadata Tablet 1
\(-\infty \text{ to } \text{"Encyclopedia:Ocelot"}\)

Metadata Tablet 2
\(\text{"Encyclopedia:Ocelot" \text{ to } \infty}\)

Table: "Adam's Table"
- Tablet 1 \(-\infty \text{ to } \text{"Things"}\)
- Tablet 2 \(\text{"Things" \text{ to } \infty}\)

Table: "Encyclopedia"
- Tablet 1 \(-\infty \text{ to } \text{"Hippo"}\)
- Tablet 2 \(\text{"Hippo" \text{ to } \text{"Ocelot"}}\)
- Tablet 3 \(\text{"Ocelot" \text{ to } \infty}\)

Table: "Foo"
- Tablet 1 \(-\infty \text{ to } \text{"Things"}\)
Tablet Data Flow

Ingest -> In-Memory Map -> Iterator Tree

Minor Compaction Scope -> Write-Ahead Log

Sorted, Indexed File -> Sorted, Indexed File

Merging / Major Compaction Scope

Query Scope -> Queries

Iterator Tree
Rfile Multi-level index
1.3 Rfile Index

- Index is an array of keys
- One key per data block
- Complete index loaded into memory when file opened
- Complete index kept in memory when writing file
- Slow load times
- Memory exhaustion
1.3 RFile
Non multi-level 9G file

Locality group : <DEFAULT>
Start block : 0
Num blocks : 182,691
Index level 0 size : 8,038,404 bytes
First key : 00000008611ae5d6 0e94:74e1 [] 1310491465052 false
Last key : 7fffffff7daec0b4e 43ee:1c1e [] 1310491475396 false
Num entries : 172,014,468
Column families : <UNKNOWN>

Meta block : BCFile.index
Raw size : 2,148,491 bytes
Compressed size : 1,485,697 bytes
Compression type : lzo

Meta block : RFile.index
Raw size : 8,038,470 bytes
Compressed size : 5,664,704 bytes
Compression type : lzo
1.4 RFile

Index Level 1

Data Block

Index Level 0

Data Block

Data Block

Data Block

Data Block
Index blocks written as data is written
• complete index not kept in memory during write
• Index block for each level kept in memory
### Multi-level 9G file

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locality group</td>
<td>&lt;DEFAULT&gt;</td>
</tr>
<tr>
<td>Start block</td>
<td>0</td>
</tr>
<tr>
<td>Num blocks</td>
<td>187,070</td>
</tr>
<tr>
<td>Index level 1</td>
<td>4,573 bytes 1 blocks</td>
</tr>
<tr>
<td>Index level 0</td>
<td>10,431,126 bytes 82 blocks</td>
</tr>
<tr>
<td>First key</td>
<td>00000000861ae5d6 0e94:74e1 [] 1310491465052 false</td>
</tr>
<tr>
<td>Last key</td>
<td>7fffffff7daec0b4e 43ee:1c1e [] 1310491475396 false</td>
</tr>
<tr>
<td>Num entries</td>
<td>172,014,468</td>
</tr>
<tr>
<td>Column families</td>
<td>&lt;UNKNOWN&gt;</td>
</tr>
</tbody>
</table>

**Meta block** : BCFFile.index

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw size</td>
<td>5 bytes</td>
</tr>
<tr>
<td>Compressed size</td>
<td>17 bytes</td>
</tr>
<tr>
<td>Compression type</td>
<td>lzo</td>
</tr>
</tbody>
</table>

**Meta block** : RFile.index

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw size</td>
<td>4,652 bytes</td>
</tr>
<tr>
<td>Compressed size</td>
<td>3,745 bytes</td>
</tr>
<tr>
<td>Compression type</td>
<td>lzo</td>
</tr>
</tbody>
</table>
Test Setup

- Files in FS cache for test (cat test.rf > /dev/null)
- Too much variability when some of file was in FS cache and some was not
- Easier to force all of file into cache than out
## Open, seek 9G file

<table>
<thead>
<tr>
<th>Multilevel Index</th>
<th>Cache</th>
<th>Avg Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>139.48 ms</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>37.55 ms</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>8.48 ms</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>3.90 ms</td>
</tr>
</tbody>
</table>
Randomly seeking 9G file

<table>
<thead>
<tr>
<th>Multilevel Index</th>
<th>Cache</th>
<th>Avg Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>2.08 ms</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>2.32 ms</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>4.33 ms</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>2.14 ms</td>
</tr>
</tbody>
</table>
Configuration

- Index block size configurable per table
- Make index block size large and it will behave like old rfile
- Enabled index cache by default in 1.4
Cache hit rate on monitor page
Fault tolerant concurrent table operations
Problematic situations

- Master dies during create table
- Create and delete table run concurrently
- Client dies during bulk ingest
- Table deleted while writing or scanning
Create table fault

• Master dies during
  • Could leave accumulo metadata in bad state
• Master creates table, but then dies before notifying client
  • If client retries, it gets table exist exception
FATE
Fault Tolerant Executor

- If process dies, previously submitted operations continue to execute on restart.
- Serializes operation in zookeeper before execution
- Master uses FATE to execute table operations
Bulk import test

- Create table with summation aggregator
- Bulk import files of +1 or -1, graph ensures final sum is 0
- Concurrently merge, split, compact, and consistency check table
- Exit point verifies 0
Adampotent

- Idempotent $f(f(x)) = f(x)$
- Adampotent $f(f'(x)) = f(x)$
  - $f'(x)$ denotes partial execution of $f(x)$
public interface Repo<T> extends Serializable {
    long isReady(long tid, T environment) throws Exception;
    Repo<T> call(long tid, T environment) throws Exception;
    void undo(long tid, T environment) throws Exception;
}

- call() returns next op, null if done
- call() and undo() must be adampotent
- undo() should clean up partial execution of isReady() or call()
FATE interface

long startTransaction();
void seedTransaction(long tid, Repo op);
TStatus waitForCompletion(long tid);
Exception getException(long tid);
void delete(long tid);
Client calling FATE on master

- Start Tx
- Seed Tx
- Wait Tx
- Delete Tx
FATE transaction states

- NEW
- IN PROGRESS
- FAILED IN PROGRESS
- SUCCESSFUL
- FAILED
public interface TStore {
    long reserve();
    Repo top(long tid);
    void push(long tid, Repo repo);
    void pop(long tid);
    Tstatus getStatus(long tid);
    void setStatus(long tid, Tstatus status);
    
    
    
    
    
}
Seeding

```java
void seedTransaction(long tid, Repo repo) {
    if (store.getStatus(tid) == NEW) {
        if (store.top(tid) == null)
            store.push(tid, repo);
        store.setStatus(tid, IN_PROGRESS);
    }
}
```
FATE transaction execution

Reserve tx -> Execute top op -> Push op -> Save Exception

Mark success -> Mark failed in progress

Pop op -> Undo top op -> Mark fail
Concurrent table ops

- Per-table read/write lock in zookeeper
- Zookeeper recipe with slight modification:
  - store fate tid as lock data
  - Use persistent sequential instead of ephemeral sequential
Concurrent delete table

- Table deleted during read or write 1.3
  - Most likely, scan hangs forever
  - Rarely, permission exception thrown (seen in test)
- In 1.4 check if table exists when:
  - Nothing in !METADATA
  - See a permission exception
Rogue threads

- Bulk import fate op pushes work to tservers
- Threads on tserver execute after fate op done
- Master deletes WID in ZK, waits until counts 0

**Tablet server worker thread**

1. Start
2. Synchronized
   - WID Exist?
3. cnt[WID]++
4. Do work
5. cnt[WID]--
6. exit
Create Table Ops

1. CreateTable
   • Allocates table id
2. SetupPermissions
3. PopulateZookeeper
   • Reentrantly lock table in isReady()
   • Relate table name to table id
4. CreateDir
5. PopulateMetadata
6. FinishCreateTable
Random walk concurrent test

- Perform all table operation on 5 well known tables
- Run multiple test clients concurrently
- Ensure accumulo or test client does not crash or hang
- Too chaotic to verify data correctness
- Found many bugs
Future Work

- Allow multiple processes to run fate operations
- Stop polling
- Hulk Smash Tolerant
Other 1.4 Features
Tablet Merging

- Splits exists forever; data is often aged off
- Admin requested, not automatic
- Merge by range or size
Merging Minor Compaction

• 1.3: minor compactions always add new files:

• 1.4: limits the total of new files:
Merging Minor Compactions

- Slows down minor compactions
- Memory fills up
- Creates back-pressure on ingest
- Prevents “unable to open enough files to scan”
Table Cloning

- Create a new table using the same read-only files
- Fast
- Testing
  - At scale, with representative data
  - Compaction with a broken iterator: no more data
- Offline and Copy
  - Create an consistent snapshot in minutes
Range Operations

- Compact Range
  - Compact all tablets that fall within a row range down to a single file
  - Useful for tail-insert indexes
  - Data purge

- Delete Range
  - Delete all the content within a row range
  - Uses split and will delete whole tablets for efficiency
  - Useful for data purge
Logical Time for Bulk Import

- Bulk files created on different machines will get different actual times (hours)
- Bulk files always contain the timestamp created by the client
- A single bulk import request can set a consistent time across all files
- Implemented using iterators
Roadmap
1.4.1 Improvements

- Snappy, LZ4
- HDFS Kerberos compatibility
- Bloom filter improvements
- Map Reduce directly over Accumulo files
- Server side row select iterator
- Bulk ingest support for map only jobs
- BigTop support
- Wikisearch example improvements
Possible 1.5 features
Performance

- Multiple namenode support
- WAL performance improvements
- In-memory map locality groups
- Timestamp visibility filter optimization
- Prefix encoding
- Distributed FATE ops
API

- Stats API
- Automatic deployment of iterators
- Tuplestream support
- Coprocessor integration
- Support other client languages
- Client session migration
Admin/Security

- Kerberos support/pluggable authentication
- Administration monitor page
- Data center replication
- Rollback/snapshot backup/recovery
Reliability

- Decentralize master operations
- Tablet server state model
Testing

- Mini-cluster test harness
- Continuous random walk testing